



# Vertex Coloring with Communication Constraints in Synchronous Broadcast Networks

Hicham Lakhlef, Michel Raynal, François Taïani

## ► To cite this version:

Hicham Lakhlef, Michel Raynal, François Taïani. Vertex Coloring with Communication Constraints in Synchronous Broadcast Networks. IEEE Transactions on Parallel and Distributed Systems, 2019, 30 (7), pp.1672-1686. 10.1109/TPDS.2018.2889688 . hal-02376726

**HAL Id: hal-02376726**

**<https://inria.hal.science/hal-02376726>**

Submitted on 22 Nov 2019

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Vertex Coloring with Communication Constraints in Synchronous Broadcast Networks

Hicham Lakhlef, Michel Raynal, François Taïani  
E-mail: hlakhlef@utc.fr, {michel.raynal, francois.taiani}@irisa.fr

**Abstract**—This paper considers distributed vertex-coloring in broadcast/receive networks suffering from conflicts and collisions. (A collision occurs when, during the same round, messages are sent to the same process by too many neighbors; a conflict occurs when a process and one of its neighbors broadcast during the same round.) More specifically, the paper focuses on multi-channel networks, in which a process may either broadcast a message to its neighbors or receive a message from at most  $\gamma$  of them. The paper first provides a new upper bound on the corresponding graph coloring problem (known as frugal coloring) in general graphs; proposes an exact bound for the problem in trees; presents a deterministic, parallel, color-optimal, collision- and conflict-free distributed coloring algorithm for trees; proves the correctness of this algorithm; and finally evaluates experimentally its performance using simulations that show our solution clearly outperforms a reference protocol for distributed TDMA slot-allocation.

**Index Terms**—Distributed algorithms, Message-passing, Broadcast/receive communication, Synchronous systems, Multi-coloring, Vertex coloring, TDMA slot allocation



## 1 INTRODUCTION

*Distributed graph coloring* is one of the fundamental problems of distributed computing research [3], [4], [7], [9]. It is particularly well adapted to situations in which resources (in the form of colors) must be allocated to processes (or nodes<sup>1</sup>). In spite of its fundamental nature, however, very few works have investigated distributed coloring in the presence of a *communication adversary*, i.e. when neighboring nodes in the communication graph are not guaranteed to be able to communicate reliably in all communication rounds.

In this paper we consider one such problem, which arises when allocating time slots in a *multi-channel* Time Division Medium Access (TDMA) wireless network. In such networks, a node with a single transceiver is unable to simultaneously receive and send at the same time (a situation known as a *conflict*), but may however simultaneously receive messages from up to  $\gamma$  neighbors without *collision* (a situation occurring when  $\gamma + 1$  or more neighbors of the same node attempt to broadcast simultaneously).

Conflicts and collisions can be avoided by allocating specific time-slots and specific channels to individual nodes, while taking into account neighboring relationships in the communication graph. In a multi-channel network, these constraints can be modeled as a specific vertex graph coloring problem, known as *frugal coloring* [12], [19], in which

the same color (representing a time slot) might be allocated up to  $\gamma$  times (the number of available channels) in a node's neighborhood. Performing this channel allocation reliably and efficiently in a distributed manner is however challenging: because channels have not been allocated yet, extra care must be taken to avoid conflicts and collisions during the allocation procedure.

Existing solutions to this problem tend to be either probabilistic [11], [13], [17], [25], or to ignore collisions and conflicts altogether [5], [10]. In this paper, we aim to improve on this situation, by seeking *deterministic* distributed solutions to frugal coloring (and hence to the TDMA slot allocation problem), that tolerate *conflicts* and *collisions* (a problem we have called *Frugal Coloring under Conflicts and Collisions*, or F3C for short). Determinism is important, as it deliver more predictable solutions, that tend to terminate faster (a point confirmed by our experimental evaluation). Robustness to conflicts and collisions is essential in practice, to allow solutions to be used in real systems in which a TDMA schedule has not been computed yet.

To solve F3C, we make the following contributions ( $\Delta$  is the tree's maximum degree, and  $\gamma$  the number of available communication channels):

- We provide an upper bound on the minimum number of colors necessary to solve F3C in a general communication graph. This bound complements existing asymptotic bounds, while putting no constraints on  $\Delta$ .
- We prove that  $\lceil \frac{\Delta}{\gamma} \rceil + 1$  colors are both necessary and sufficient to solve F3C in trees.
- We propose and prove the correctness of a *deterministic, color-optimal, parallel, collision- and conflict-free, distributed protocol* that solves F3C on trees in  $O(d \lceil \frac{\Delta}{\gamma} \rceil)$  steps,

• H. Lakhlef is with the Université Technologique de Compiègne (UTC), France; M. Raynal and F. Taïani are with Univ Rennes, CNRS, Inria, IRISA (F-35000 Rennes); M. Raynal is also Chair Professor at the Polytechnic University of Hong Kong.  
• A preliminary version of this paper was published in [15].

1. We use both words interchangeably, with a preference for *process* when describing an algorithmic behavior.

where  $d$  is the tree's depth.

In the following, we first present some background and motivate the F3C problem (Sec. 2). We then describe the underlying system model (Sec. 3), before formally defining the F3C problem (Sec. 4). We then present an upper bound on the minimal number of colors  $K$  necessary to solve F3C in general graphs, and a lower bound on  $K$  in trees (Sec. 5). We then present our algorithm which solves F3C in trees (Sec. 6) and prove it (Sec. 7). We conclude in Section 9, and provide some prospective remarks.

## 2 BACKGROUND AND PROBLEM

### 2.1 Distributed coloring in wireless networks

A large number of works [3], [5], [9], [10], [11], [13], [17] have proposed distributed vertex coloring algorithms applicable to wireless networks. Unfortunately, the coloring protocols proposed so far are either not robust to both conflicts and collisions [5], [10]; rely on probabilistic procedures to eventually converge to a solution with high probability [11], [13], [17], [25]; or adopt a sequential approach [9], in which one single process is communicating at a time, a particularly slow procedure. In terms of coloring, these protocols do not focus on multi-channel networks in which the same slot/color might be allocated several times among collision-prone neighbors.

Let us note, for the sake of completeness, that the multi-channel allocation problem can also be modeled using other types of coloring, such as *edge-coloring* [26] (when focusing on wireless-links rather than on broadcast operations as we do), *multicoloring* [20] (when ignoring conflicts), or *t-coloring* [22] (when investigating cross-channel interference). These are crucial problems and contributions in their own right; they lie, however, out of the scope of the present work.

### 2.2 Distributed Frugal Coloring

A  $\gamma$ -frugal coloring [8], [12], [18], [19] is a particular type of graph coloring in which (i) two neighboring nodes must receive different colors (thus avoiding conflicts when mapping colors to time slots), and (ii) the same color might not be used more than  $\gamma$  times in a node's neighborhood (thus avoiding collisions in an TDMA network using  $\gamma$  channels).

Several important theoretical results exist on this type of coloring, but these are mostly limited to the asymptotic behavior of the minimal number of colors required to color a graph. *Asymptotic* means that these results typically hold for sufficiently large values of  $\Delta$ , the maximum degree of the graph. For instance, Molloy and Reed [19] showed that if a graph's maximum degree  $\Delta$  is sufficiently large, a  $(50 \ln \Delta / \ln \ln \Delta)$ -frugal coloring exists with  $\Delta + 1$  colors (and that this value is in fact optimal for this frugality value in that some graphs cannot be colored with fewer colors). Hind *et al.* [12] showed that (i) for a sufficiently large  $\Delta$  and any  $\gamma \geq 1$ , a  $\gamma$ -frugal coloring exists with at most  $\max \left\{ (\gamma + 1)\Delta, \left\lceil e^3 \frac{\Delta^{1+1/\gamma}}{\gamma} \right\rceil \right\}$  colors, and that (ii) for arbitrary large  $\Delta$  there is a graph with maximum degree  $\Delta$  that cannot be  $\gamma$ -frugally colored with fewer than  $\frac{\Delta^{1+1/\gamma}}{2\gamma}$  colors. In the same vein, Molloy and Reed showed that if a graph's maximum degree  $\Delta$  is larger than  $\gamma^\gamma$  then a  $\gamma$ -frugal coloring exists with  $16\Delta^{1+\frac{1}{\gamma}}$  colors [18].

These results unfortunately do not translate easily to wireless networks. Assuming  $\gamma = 7$  channels for instance, the last result states the existence of a coloring (or schedule) with  $\lceil 16 \times (7^7)^{1+\frac{1}{7}} \rceil = 92,236,816$  colors (i.e. slots). Assuming a slot duration of 50 ms, this corresponds to an overall schedule of 53 days, an unpractical duration for many wireless applications. This growth is also over-exponential, rising to 1242 days (or 3.4 years) for only  $\gamma = 8$  channels, 88 years for 9 channels, and 2,536 years for 10 channels.

Chung, Pettie and Su have proposed distributed algorithms [8] that realize some of the above frugal coloring results. Unfortunately, besides the unpracticality of the resulting coloring as a slot schedule, these algorithms have so far been probabilistic (they converge with high probability), and do not take into account collisions or conflicts in their distributed execution.

### 2.3 Problem and contributions

In contrast to the above results, we take a much more hands-on perspective in this work. We focus on the *distributed deterministic frugal-coloring of realistic graphs*, under *adversarial communication constraints* capturing conflicts and collisions, in a synchronous  $\gamma$ -channel wireless network. This problem, which we have termed *Frugal Coloring under Conflicts and Collisions* (F3C for short), has so far, and to the best of our knowledge, not yet been investigated.

Rather than targeting asymptotic results, we do not make any assumption on the maximum degree  $\Delta$  of the graph, so that our results remain applicable to any network. We target *deterministic* algorithms in order to provide strong guarantees of rapid convergence in practical cases. Our overall goal is to solve the F3C problem for a given number of channels  $\gamma$  with the smallest possible number of colors  $K$  (thus reducing the overall size of the resulting slot allocation schedule), and in few synchronous rounds (avoiding whenever possible sequential solutions).

## 3 SYNCHRONOUS BROADCAST/RECEIVE MODEL

### 3.1 Processes, initial knowledge, and graph

The system model consists of  $n$  sequential processes denoted  $p_1, \dots, p_n$ . These processes are organized in a communication graph that is connected and undirected, and reflects the limited communication range of wireless communications. When considering a process  $p_i$ ,  $1 \leq i \leq n$ , the integer  $i$  is called its index. Indexes are not known to the processes. They are only a notation convenience used to distinguish processes and their local variables.

Each process  $p_i$  has an identity  $id_i$ , which is known only to itself and its neighbors (processes at distance 1 from it). The constant  $neighbors_i$  is a local set, known only to  $p_i$ , which contains the identities of  $p_i$ 's neighbors (and only them). In order for a process  $p_i$  not to confuse its neighbors, it is assumed that no two processes at distance less than or equal to 2 have the same identity. Two processes lying at a distance greater than 2 may however use the same identifier.

$\Delta_i$  denotes the degree of process  $p_i$  (i.e.  $|neighbors_i|$ ) and  $\Delta$  denotes the maximal degree of the graph ( $\max\{\Delta_1, \dots, \Delta_n\}$ ). While each process  $p_i$  knows  $\Delta_i$ , no



Table 1  
Notations used when manipulating multisets, balls, and spheres

notation	meaning
$set(M)$	the underlying set of $M$ , i.e. the set of elements present at least once in $M$
$\mathbf{1}_M(x)$	the multiplicity of an element $x$ in the multiset $M$
$ M $	the cardinality of $M$
$M \otimes m$	the $m$ -multiple of $M$
$A \uplus B$	the multiset union of the multisets $A$ and $B$
$A \cup B$	the generalized set union of $A$ and $B$
$A \setminus B$	the generalized set difference of $A$ and $B$
$A \cap B$	the generalized set intersection of the multisets $A$ and $B$
$f(S)$	image of the set $S$ by the function $f$
$f[M]$	multi-image of the multiset $M$ by the function $f$
$S_r^p$	the sphere of radius $r$ centered on process $p$
$\mathcal{B}_r^p$	the ball of radius $r$ centered on process $p$

colors of two neighbors are simply requested to be disjoint), and  $\text{F3C}(n, 1, K, 1)$  captures the classical distance-2 coloring problem (vertices at distance  $\leq 2$  have different colors).

The reader can easily see that  $\text{F3C}(n, \gamma, K, \geq 1)$  captures the general coloring problem informally stated in the introduction. Once a process  $p_i$  has been assigned a set of colors  $colors_i$ , at the application programming level, it is allowed to broadcast a message to neighbors at the rounds (time slots) corresponding to the values of  $CLOCK$  such that  $(CLOCK \bmod K) \in colors_i$ .

## 5 IMPOSSIBILITY, LOWER BOUND, AND UPPER BOUND IN TREES AND GENERAL GRAPHS

### 5.1 An impossibility result

Generalizing a remark from [14] this section presents a lower bound on  $K$ : neither  $\text{F3C}(n, \gamma, K, 1)$ , nor  $\text{F3C}(n, \gamma, K, \geq 1)$ , can be solved for  $K \leq \lceil \frac{\Delta}{\gamma} \rceil$ . The next sections will present an algorithm solving  $\text{F3C}(n, \gamma, K, \geq 1)$  in trees in the synchronous model described in Section 3, and a proof of it. This algorithm is such  $K = \lceil \frac{\Delta}{\gamma} \rceil + 1$ , and is consequently optimal with respect to the total number of colors.

**Theorem 1.** *Neither  $\text{F3C}(n, \gamma, K, 1)$ , nor  $\text{F3C}(n, \gamma, K, \geq 1)$  can be solved when  $K \leq \lceil \frac{\Delta}{\gamma} \rceil$ .*

The proof of Theorem 1 follows from a simple counting argument on the number of colors required to color the neighborhood of a process with  $\Delta$  neighbors. It is provided in the Appendix for space reasons.

Note that the resulting lower bound  $K \geq \lceil \frac{\Delta}{\gamma} \rceil + 1$  necessary to solve F3C is valid for *any* graph. This contrasts for instance with the lower bound  $\frac{\Delta^{1+1/\gamma}}{2\gamma}$  proposed by Hind *et al.* [12], which holds for *some* graph of degree  $\Delta$ .

### 5.2 $\text{F3C}(n, \gamma, K, 1)$ in a General Network

**Theorem 2.**  $\Delta + \lfloor \frac{\Sigma}{\gamma} \rfloor + 1$  colors are sufficient to solve  $\text{F3C}(n, \gamma, K, 1)$  in a general graph, where  $\Sigma$  is the size of the largest sphere of radius 2 in  $G$ , i.e.,

$$\Sigma = \max_{p \in \Pi} |\{q \in \Pi \mid \text{dist}(p, q) = 2\}|,$$

and  $\text{dist}(p, q)$  represents the hop-distance between two processes in  $G$ .

The above upper bound does not put any constraint on the maximum degree  $\Delta$  of the graph, contrarily to earlier asymptotic results on  $\gamma$ -frugal coloring [12], [18], [19], [8]. It also uses  $\Sigma$ , the size of the largest sphere of radius 2. It is in that respect complementary to these earlier bounds, which only rely on  $\Delta$ .

**Proof** The proof of Theorem 2 relies on the analysis of Algorithm 1, which implements a classical greedy sequential procedure that solves  $\text{F3C}(n, \gamma, K, 1)$  in a centralized shared memory model on an arbitrary connected graph  $G$ . Algorithm 1 and the proof use multisets<sup>2</sup> with the following notations (summarized in Table 1):

- $set(M)$  is the underlying set of  $M$ , the set of elements present at least once in  $M$ .
- $\mathbf{1}_M(x)$  is the multiplicity of an element  $x$  in the multiset  $M$ . By construction we have

$$\mathbf{1}_M(x) \geq 1 \iff x \in set(M). \quad (1)$$

- $|M|$  is the cardinality of  $M$ ; in particular if  $M$  has finite cardinality (which is the case of all sets and multisets used in the following)

$$|M| = \sum_{x \in set(M)} \mathbf{1}_M(x). \quad (2)$$

- $M \otimes m$ , where  $m$  is an integer, is the  $m$ -multiple of  $M$ , defined by

$$\mathbf{1}_{M \otimes m}(x) = \mathbf{1}_M(x) \times m. \quad (3)$$

- $A \uplus B$  is the multiset union of the multisets  $A$  and  $B$ , defined by

$$\mathbf{1}_{A \uplus B}(x) = \mathbf{1}_A(x) + \mathbf{1}_B(x). \quad (4)$$

- $A \cup B$  is the generalized set union of  $A$  and  $B$ , defined by

$$\mathbf{1}_{A \cup B}(x) = \max\{\mathbf{1}_A(x), \mathbf{1}_B(x)\}. \quad (5)$$

- $A \setminus B$  is the generalized set difference of  $A$  and  $B$ , defined by

$$\mathbf{1}_{A \setminus B}(x) = \max\{0, \mathbf{1}_A(x) - \mathbf{1}_B(x)\}. \quad (6)$$

- $A \cap B$  is the generalized set intersection of the multisets  $A$  and  $B$ , defined by

$$\mathbf{1}_{A \cap B}(x) = \min\{\mathbf{1}_A(x), \mathbf{1}_B(x)\}. \quad (7)$$

We consider a set  $S$  as a special case of a multiset in which all elements of  $S$  have a multiplicity of 1:

$$x \in S \iff \mathbf{1}_S(x) = 1.$$

Algorithm 1 iterates sequentially over all processes in the system (line 1). For each process  $p_i$ , the algorithm enumerates the colored tokens that are no longer available to  $p_i$ . First  $\gamma$  colored tokens are counted in the multiset *neighbors\_toks* for each neighbor of  $p_i$  (line 2): in order to respect conflict freedom, these colors cannot be used to color  $p_i$ . The multiset *siblings\_toks* at line 3 counts the number of times a color is used at distance 2 of  $p_i$ : in order to respect  $\gamma$ -collision freedom, these colors can only be used for  $p_i$  if

2. Differently from a set, a *multiset* (also called a *bag*), can contain several times the same element. Hence, while  $\{a, b, c\}$  and  $\{a, b, a, c, c, c\}$  are the same set, they are different multisets.

```

1 Init: Process colors are initialized to  $\perp$ :  $\forall p_i \in \Pi : color_i = \perp$ .
2 for each  $p_i$  in  $\Pi$  do
3    $neighbors\_toks \leftarrow \{color_j : p_j \in neighbors_i\} \otimes \gamma$ ;
4    $siblings\_toks \leftarrow \bigcup_{p_j \in neighbors_i} \biguplus_{p_k \in neighbors_j} \{color_k\}$ ;
5    $available\_toks \leftarrow \mathbb{N} \otimes \gamma \setminus (neighbors\_toks \cup siblings\_toks)$ ;
6    $color_i \leftarrow \min(set(available\_toks))$ .

```

Algorithm 1: Sequential F3C( $n, \gamma, K, 1$ ) for a graph, based on a greedy strategy

they have been used less than  $\gamma$  times in all neighborhoods  $neighbors_j$  to which  $p_i$  belongs. Finally,  $neighbors\_toks$  and  $siblings\_toks$  are removed from  $\mathbb{N} \otimes \gamma$ , the multiset in which all natural numbers are present  $\gamma$  times, and the lowest remaining available color is allocated to  $p_i$  (line 4). During this iteration step, some of  $p_i$ 's neighbors and neighbors of neighbors might not have been allocated a color yet: their color is formally equal to  $\perp$ , and eliminated at line 4, with no impact of the final value of  $color_i$ .

For this proof we introduce the notion of the *multi-image* of a *multiset* by a function, which extends that of the image of a set by a function. If  $M$  is a multiset and  $f$  a function defined over  $set(M)$ , then  $f[M]$  is the multiset defined by<sup>3</sup>:

$$set(f[M]) = f(set(M)), \quad (8)$$

$$\forall y \in set(f[M]) : \mathbf{1}_{f[M]}(y) = \sum_{x \in f^{-1}(y)} \mathbf{1}_M(x). \quad (9)$$

where  $f(set(M))$  is the traditional image<sup>4</sup> by  $f$  of the set  $set(M)$ .

This definition implies the following properties:

$$|f[M]| = |M|, \quad (10)$$

$$f[M \otimes m] = f[M] \otimes m, \quad (11)$$

$$f[M] \uplus f[N] = f[M \uplus N], \quad (12)$$

$$f[M] \cup f[N] \subseteq f[M \cup N]. \quad (13)$$

To prove the theorem, we show that the variable  $color_i$  allocated in each iteration is smaller or equal to  $\Delta + \lfloor \frac{\Sigma}{\gamma} \rfloor$ . To this aim we first compute an upper bound of the cardinal of  $(neighbors\_toks \cup siblings\_toks)_{|\mathbb{N}}$ , where  $M_{|S}$  denotes the restriction of the multiset  $M$  to the elements of the set  $S$ .

Let us note  $col_i$  the function that associates each process to its coloring at the start of iteration  $i$

$$\begin{aligned} col_i : \Pi &\mapsto \mathbb{N} \cup \{\perp\}, \\ p_j &\mapsto color_j. \end{aligned} \quad (14)$$

Note that by construction,  $col_i(p_i) = \perp$ .

In the following, we note  $\mathcal{S}_r^p$  the sphere of radius  $r$  centered on process  $p$ :

$$\mathcal{S}_r^p = \{q \in \Pi \mid dist(p, q) = r\}.$$

With this definition we have  $\mathcal{S}_0^{p_i} = \{p_i\}$ ,  $\mathcal{S}_1^{p_i} = neighbors_i$ ,  $\Delta = \max_{p \in \Pi} |\mathcal{S}_1^p|$ , and  $\Sigma = \max_{p \in \Pi} |\mathcal{S}_2^p|$ . We also define

3. An equivalent and more compact definition could also be  $f[M] = \biguplus_{x \in M} (\{f(x)\} \otimes \mathbf{1}_M(x))$ .

4. Note that in the general case, for a set  $S$ ,  $f(S) \subseteq f[S]$ , i.e. the image of  $S$  by  $f$  is included in its multi-image by  $f$ , but  $f(S)$  and  $f[S]$  are different.  $f(S) = f[S]$  only holds for functions that are injective on  $S$ .

the ball of radius  $r$  centered on process  $p$  as:  $\mathcal{B}_r^p = \{q \in \Pi \mid dist(p, q) \leq r\}$ .

Using the above definitions and properties, we have

$$neighbors\_toks = col_i(neighbors_i) \otimes \gamma, \quad (15)$$

$$\subseteq col_i[neighbors_i] \otimes \gamma, \quad (16)$$

$$\subseteq col_i[\mathcal{S}_1^{p_i}] \otimes \gamma, \quad (17)$$

$$\subseteq col_i[\mathcal{S}_1^{p_i} \otimes \gamma]. \quad (18)$$

Similarly, we also have

$$\begin{aligned} siblings\_toks &= \bigcup_{p_j \in neighbors_i} \biguplus_{p_k \in neighbors_j} col_i[\{p_k\}], \quad (19) \\ &= \bigcup_{p_j \in neighbors_i} col_i[\biguplus_{p_k \in neighbors_j} \{p_k\}], \quad (20) \\ &= \bigcup_{p_j \in neighbors_i} col_i[neighbors_j], \quad (21) \end{aligned}$$

$$\subseteq col_i[\bigcup_{p_j \in neighbors_i} neighbors_j], \quad (22)$$

$$\subseteq col_i[\mathcal{B}_2^{p_i}]. \quad (23)$$

Combining (18) and (23) yields:

$$\begin{aligned} neighbors\_toks \cup siblings\_toks &\subseteq col_i[\mathcal{S}_1^{p_i} \otimes \gamma] \cup col_i[\mathcal{B}_2^{p_i}], \quad (24) \\ &\subseteq col_i[\mathcal{S}_1^{p_i} \otimes \gamma \cup \mathcal{B}_2^{p_i}], \quad (25) \\ &\subseteq col_i[\mathcal{S}_1^{p_i} \otimes \gamma \cup \mathcal{S}_0^{p_i} \cup \mathcal{S}_1^{p_i} \cup \mathcal{S}_2^{p_i}]. \quad (26) \end{aligned}$$

Both  $neighbors\_toks$  and  $siblings\_toks$  might contain bottom values ( $\perp$ ), which we need to eliminate in order to reason about the smallest integer remaining in  $available\_toks$  at line 5. To this aim, we introduce the restriction of the multiset  $X$  to values of  $S$ , noted  $X_{|S}$ , and defined as

$$\mathbf{1}_{X_{|S}}(x) = \begin{cases} \mathbf{1}_X(x) & \text{if } x \in S \\ 0 & \text{if } x \notin S \end{cases}$$

Let us note the following two properties of multisets, where  $M$  and  $N$  are multisets,  $m$  and  $n$  are positive integers, and  $S$  is a set:

$$M \otimes m \cup M \otimes n = M \otimes \max(m, n), \quad (27)$$

$$f[N] \cap S = \emptyset \Rightarrow f[M]_{|S} \subseteq f[M \setminus N]. \quad (28)$$

Applying the above two properties to (26) yields:

$$\begin{aligned} (neighbors\_toks \cup siblings\_toks)_{|\mathbb{N}} &\subseteq \left( col_i[\mathcal{S}_1^{p_i} \otimes \gamma \cup \mathcal{S}_1^{p_i} \cup \mathcal{S}_2^{p_i} \cup \mathcal{S}_0^{p_i}] \right)_{|\mathbb{N}}, \quad (29) \\ &\subseteq \left( col_i[\mathcal{S}_1^{p_i} \otimes \gamma \cup \mathcal{S}_2^{p_i} \cup \mathcal{S}_0^{p_i}] \right)_{|\mathbb{N}}, \quad (30) \\ &\subseteq \left( col_i[\mathcal{S}_1^{p_i} \otimes \gamma \cup \mathcal{S}_2^{p_i}] \right)_{|\mathbb{N}}. \quad (31) \end{aligned}$$

Taking the cardinal we have

$$\begin{aligned} |(neighbors\_toks \cup siblings\_toks)_{|\mathbb{N}}| &\leq |(col_i[\mathcal{S}_1^{p_i} \otimes \gamma \cup \mathcal{S}_2^{p_i}])_{|\mathbb{N}}|, \quad (32) \\ &\leq |col_i[\mathcal{S}_1^{p_i} \otimes \gamma \cup \mathcal{S}_2^{p_i}]|, \quad (33) \\ &\leq |\mathcal{S}_1^{p_i} \otimes \gamma \cup \mathcal{S}_2^{p_i}|, \quad (34) \\ &\leq |\mathcal{S}_1^{p_i} \otimes \gamma| + |\mathcal{S}_2^{p_i}|, \quad (35) \\ &\leq \Delta \times \gamma + \Sigma. \quad (36) \end{aligned}$$

$color_i$  at line 5 is computed in such a way that

$$\forall k < color_i : \mathbf{1}_{available\_toks}(k) = 0, \quad (37)$$

hence,

$$\forall k < color_i : \mathbf{1}_{neighbors\_toks \cup siblings\_toks}(k) \geq \gamma, \quad (38)$$

from which we derive,

$$|(neighbors\_toks \cup siblings\_toks)_{|\mathbb{N}}| \geq \gamma \times color_i, \quad (39)$$

and so with (36)

$$\gamma \times color_i \leq \Delta \times \gamma + \Sigma, \quad (40)$$

$$color_i \leq \Delta + \frac{\Sigma}{\gamma}, \quad (41)$$

$$color_i \leq \Delta + \left\lceil \frac{\Sigma}{\gamma} \right\rceil. \quad (42)$$

This shows that the algorithm allocates only colors in the range  $[0, \dots, \Delta + \lceil \frac{\Sigma}{\gamma} \rceil]$ , which concludes the proof of the theorem.  $\square_{Theorem 2}$

The bound of Theorem 2 is realized by the complete graph (in which  $\Sigma = 0$ ), but is in general not tight, as the results we present for trees later on demonstrate (Algorithm 2 and Theorem 4 in Section 6 and 7). Algorithm 1 can also serve as the basis for a naive sequential (and hence C2-free) distributed coloring procedure, that would propagate the state of the entire graph coloring from process to process, incurring messages with  $O(n)$  length, and executing in  $O(n)$  steps.

The subsequent contributions of this paper show that a C2-free parallel solution exists for trees that uses: (i) messages of length at most  $O(\Delta)$ ; (ii)  $O(d \lceil \frac{\Delta}{\gamma} \rceil)$  steps; and (iii) an optimal number of colors. An open problem remains however whether F3C( $n, \gamma, K, 1$ ) can be solved deterministically with a *parallel* algorithm in an arbitrary graph in a system that suffers conflicts and  $\gamma$ -collisions.

### 5.3 A necessary and sufficient condition for F3C

Let F3C( $n, \gamma, \lceil \frac{\Delta}{\gamma} \rceil + 1, > 1$ ) denote the problem F3C( $n, \gamma, \lceil \frac{\Delta}{\gamma} \rceil + 1, \geq 1$ ) where at least one process obtains more than one color.

**Theorem 3.** F3C( $n, \gamma, \lceil \frac{\Delta}{\gamma} \rceil + 1, > 1$ ) can be solved on a tree of maximal degree  $\Delta$ , iff

$$\exists i : \lceil \frac{\Delta}{\gamma} \rceil > \max \left( \left\{ \left\lceil \frac{\Delta_i}{\gamma} \right\rceil \right\} \cup \left\{ \left\lfloor \frac{\Delta_j}{\gamma} \right\rfloor \mid p_j \in neighbors_i \right\} \right).$$

To not overload the presentation the formal proof of this theorem is given in the Appendix.

## 6 SOLVING F3C( $n, \gamma, K, \geq 1$ ) IN A TREE

The algorithm presented in this section uses as a skeleton a parallel traversal of a tree [24]. Such a traversal is implemented by control messages that visit all the processes, followed by a control flow that returns to the process that launched the tree traversal.

As claimed previously, Algorithm 2 is a  $K$ -optimal, parallel, C2 $\gamma$ -free algorithm that solves F3C( $n, \gamma, \lceil \frac{\Delta}{\gamma} \rceil + 1, \geq 1$ ) using messages of length at most  $O(\Delta)$  in  $O(d \lceil \frac{\Delta}{\gamma} \rceil)$  steps.

It assumes that a single process initially receives an external message START(), which dynamically defines it as the root of the tree. This message and the fact that processes at distance smaller or equal to 2 do not have the same identity provide the initial asymmetry from which a deterministic coloring algorithm can be built. The reception of the message START() causes the receiving process (say  $p_r$ ) to simulate the reception of a fictitious message COLOR(), which initiates the sequential traversal.

### 6.1 Messages

The algorithm uses two types of message, denoted COLOR() and TERM().

- The messages COLOR() implement a control flow visiting in parallel the processes of the tree from the root to the leaves. Each of them carries three values, denoted *sender*, *cl\_map*, and *max\_cl*.
  - *sender* is the identity of the sender of the message. If it is the first message COLOR() received by a process  $p_i$ , *sender* defines the parent of  $p_i$  in the tree.
  - *cl\_map* is a dictionary data structure with one entry for each element in  $neighbors_x \cup \{id_x\}$ , where  $p_x$  is the sender of the message COLOR(). *cl\_map*[ $id_x$ ] is the set of colors currently assigned to the sender and, for each  $id_j \in neighbors_x$ , *cl\_map*[ $id_j$ ] is the set of colors that  $p_x$  proposes for  $p_j$ .
  - *max\_cl* is an integer defining the color domain used by the sender, namely the color set  $\{0, 1, \dots, (max\_cl - 1)\}$ . Each child  $p_i$  of the message sender will use the color domain defined by  $\max(max\_cl, \sigma_i)$  to propose colors to its own children ( $\sigma_i$  is defined below). Moreover, all the children of the sender will use the same slot span  $\{0, 1, \dots, (max\_cl - 1)\}$  to schedule their message broadcasts. This ensures that their message broadcasts will be collision-free<sup>5</sup>.
- The messages TERM() propagate the return of the control flow from the leaves to the root. Each message TERM() carries two values: the identity of the destination process (as this message is broadcast, this allows any receiver to know whether it should process this message), and the identity of the sender.

### 6.2 Local variables

Each process  $p_i$  manages the following local variables. The constant  $\Delta_i = |neighbors_i|$  is the degree of  $p_i$ , while the constant  $\sigma_i = \lceil \frac{\Delta_i}{\gamma} \rceil + 1$  is the number of colors needed to color the star graph made up of  $p_i$  and its neighbors.

- *state<sub>i</sub>* (initialized to 0) is used by  $p_i$  to manage the progress of the tree traversal. Each process traverses five different states during the execution of the algorithm. States 1 and 3 are active states: a process in state 1 broadcasts a COLOR() message to its neighbors, while a process in state 3 broadcasts a message TERM() which has a meaning only for its parent. States 0 and 2 are

5. As we will see, conflicts are prevented by the message exchange pattern imposed by the algorithm.

waiting states in which a process listens on the broadcast channels but cannot send any message. Finally, state 4 identifies local termination.

- $parent_i$  stores the identity of the process  $p_j$  from which  $p_i$  receives a message  $COLOR()$  for the first time (hence  $p_j$  is the parent of  $p_i$  in the tree). The root  $p_r$  of the tree, defined by the reception of the external message  $START()$ , is the only process such that  $parent_r = id_r$ .
- $colored_i$  is a set containing the identities of the neighbors of  $p_i$  that have been colored.
- $to\_color_i$  is the set of neighbors to which  $p_i$  must propagate the coloring (network traversal).
- $color\_map_i[neighbors_i \cup \{id_i\}]$  is a dictionary data structure where  $p_i$  stores colors of its neighbors in  $color\_map_i[neighbors_i]$ , and its own colors in  $color\_map_i[id_i]$ ;  $colors_i$  is used as a synonym of  $color\_map_i[id_i]$ .
- $max\_cl_i$  defines both the color domain from which  $p_i$  can color its children, and the time slots (rounds) at which its children will be allowed to broadcast.
- $slot\_span_i$  is set to the value  $max\_cl$  carried by the message  $COLOR()$  received by  $p_i$  from its parent. As this value is the same for all the children of its parent, they will use the same slot span to define the slots during which each child will be allowed to broadcast messages.

### 6.3 Initial state

In its initial state ( $state_i = 0$ ), a process  $p_i$  waits for a message  $COLOR()$ . As said previously, a single process receives the external message  $START()$ , which defines it at the root process. It is assumed that  $CLOCK = 0$  when a process receives this message. When this happens, the corresponding process  $p_i$  simulates the reception of the message  $COLOR(id_i, cl\_map, \sigma_i)$  where  $cl\_map[id_i]$  defines its color, namely,  $(CLOCK + 1) \bmod \sigma_i$  (lines 2-4). As a result, at round number 1, the root broadcasts a message  $COLOR()$  to its children (line 33).

### 6.4 Algorithm: reception of a message $COLOR()$

When a process  $p_i$  receives a message  $COLOR()$  for the first time, it is visited by the network traversal, and must consequently (a) obtain an initial color set, and (b) propagate the network traversal, if it has children. The processing by  $p_i$  of this first message  $COLOR(sender, cl\_map, max\_cl)$  is done at lines 7-29. First,  $p_i$  saves the identity of its parent (the sender of the message) and its proposed color set (lines 7-8), initializes  $colored_i$  to  $\{sender\}$ , and  $to\_color_i$  to its other neighbors (lines 9-10). Then  $p_i$  obtains a color set proposal from the dictionary  $cl\_map$  carried by the message (line 11), computes the value  $max\_cl_i$  from which its color palette will be defined, and saves the value  $max\_cl$  carried by the message  $COLOR()$  in the local variable  $slot\_span_i$  (line 12). Let us remind that the value  $max\_cl_i$  allows  $p_i$  to know the color domain used up to now, and the rounds at which it will be able to broadcast messages (during the execution of the algorithm) in a collision-free way.

Then, the behavior of  $p_i$  depends on the value of  $to\_color_i$ . If  $to\_color_i$  is empty,  $p_i$  is a leaf, and there is no more process to color from it. In this case,  $p_i$  proceeds to state 3 (line 29).

```

1 Init:  $\sigma_i = \lceil \frac{\Delta_i}{\gamma} \rceil + 1$ ;  $state_i \leftarrow 0$ ;  $colors_i \leftarrow \emptyset$ ;  $colors_i$  stands for
    $color\_map_i[id_i]$ .
2 when  $START()$  is received do  $\triangleright$  A single proc. receives this message.
3    $p_i$  executes lines 6-30 as if it received
    $COLOR(id_i, cl\_map, \sigma_i)$ 
4   where  $cl\_map[id_i] = \{(CLOCK + 1) \bmod \sigma_i\}$ .
5 when  $COLOR(sender, cl\_map, max\_cl)$  is received do
6   if first message  $COLOR()$  received then
7      $parent_i \leftarrow sender$ ;
8      $color\_map_i[parent_i] \leftarrow cl\_map[sender]$ ;
9      $colored_i \leftarrow \{sender\}$ ;
10     $to\_color_i \leftarrow neighbors_i \setminus \{sender\}$ ;
11     $color\_map_i[id_i] \leftarrow cl\_map[id_i]$ ;  $\triangleright$  Synonym of  $colors_i$ 
12     $max\_cl_i \leftarrow \max(max\_cl, \sigma_i)$ ;  $slot\_span_i \leftarrow max\_cl$ ;
13    if ( $to\_color_i \neq \emptyset$ ) then  $\triangleright$  next lines:  $tokens_i$  is a multiset.
       $tokens_i \leftarrow \{ \gamma \text{ tokens with color } x, \forall x \in ([0, \dots, (max\_cl_i - 1)] \setminus colors_i) \}$ 
       $\setminus \{ 1 \text{ token with color } z, \forall z \in color\_map_i[parent_i] \}$ ;
14    while ( $|tokens_i| < |to\_color_i|$ ) do
15      if ( $|colors_i| > 1$ ) then
16        let  $cl \in colors_i$ ; suppress  $cl$  from  $colors_i$ ;
17        add  $\gamma$  tokens colored  $cl$  to  $tokens_i$ ;
18      else
19        let  $cl$  be the maximal color
20        in  $color\_map_i[parent_i]$ ;
21        add one token colored  $cl$  to  $tokens_i$ ;
22         $color\_map_i[parent_i] \leftarrow$ 
23         $color\_map_i[parent_i] \setminus \{cl\}$ ;
24      Extract  $|to\_color_i|$  non-empty non-intersecting
      multisets  $tk[id]$  (where  $id \in to\_color_i$ ) from  $tokens_i$ 
      such that no  $tk[id]$  contains several tokens with the
      the same color;
25      for each  $id \in to\_color_i$  do
26         $color\_map_i[id] \leftarrow \{ \text{colors of the tokens in}$ 
27         $tk[id] \}$ ;
28       $state_i \leftarrow 1$ ;  $\triangleright p_i$  has children
29    else  $state_i \leftarrow 3$ ;  $\triangleright p_i$  is a leaf
30    else  $color\_map_i[id_i] \leftarrow color\_map_i[id_i] \cap cl\_map[id_i]$ .
31 when ( $(CLOCK \bmod slot\_span_i) \in colors_i$ )  $\wedge$  ( $state_i \in \{1, 3\}$ )
    do
32   case ( $state_i = 1$ )
33      $\hookrightarrow$  bcast  $COLOR(id_i, color\_map_i, max\_cl_i)$ ;  $state_i \leftarrow 2$ ;
34   case ( $state_i = 3$ )  $\triangleright p_i$ 's subtree is done
35      $\hookrightarrow$  bcast  $TERM(parent_i, id_i)$ ;  $state_i \leftarrow 4$ .
36 when  $TERM(dest, id)$  is received do
37   if ( $dest \neq id_i$ ) then discard the message (skip lines 38-41);
38    $colored_i \leftarrow colored_i \cup \{id\}$ ;
39   if  $colored_i = neighbors_i$  then
40     if  $parent_i = id_i$  then the root  $p_i$  claims termination;
41     else  $state_i \leftarrow 3$ .

```

Algorithm 2:  $C2\gamma$ -free algorithm solving  $CCMC(n, \gamma, \lceil \frac{\Delta}{\gamma} \rceil + 1, \geq 1)$  in tree networks (code for  $p_i$ )

If  $to\_color_i$  is not empty,  $p_i$  has children. It has consequently to propose a set of colors for each of them, and save these proposals in its local dictionary  $color\_map_i[neighbors_i]$ . To this end,  $p_i$  computes first the domain of colors it can use, namely, the set  $\{0, 1, \dots, (max\_cl_i - 1)\}$ , and considers that each of these colors  $c$  is represented by  $\gamma$  tokens colored  $c$ . Then, it computes the multiset, denoted  $tokens_i$ , containing all the



colored tokens it can use to build a color set proposal for each of its children (line 14). The multiset  $tokens_i$  is initially made up of all possible colored tokens, from which are suppressed (a) all tokens associated with the colors of  $p_i$  itself, and, (b) one colored token for each color in  $color\_map_i[parent_i]$  (this is because, from a coloring point of view, its parent was allocated one such colored token for each of its colors).

Then,  $p_i$  checks if it has enough colored tokens to allocate at least one colored token to each of its children (assigning thereby the color of the token to the corresponding child). If the predicate  $|tokens_i| \geq |to\_color_i|$  is satisfied,  $p_i$  has enough colored tokens and can proceed to assign set of colors to each of its children (lines 25-27). Differently, if the predicate  $|tokens_i| < |to\_color_i|$  is satisfied,  $p_i$  has more children than available colored tokens. It must therefore find more colored tokens. To do that, if  $colors_i$  (i.e.,  $color\_map_i[id_i]$ ) has more than one color,  $p_i$  suppresses one color from  $colors_i$ , adds the  $\gamma$  associated colored tokens to the multiset  $tokens_i$  (lines 16-18), and re-enters the “while” loop (line 15). If  $colors_i$  has a single color, this color cannot be suppressed from  $colors_i$ . In this case,  $p_i$  considers the color set of its parent ( $color\_map_i[parent_i]$ ), takes the maximal color of this set, suppresses it from  $color\_map_i[parent_i]$ , adds the associated colored token to the multiset  $tokens_i$  (lines 19-24), and—as before—re-enters the “while” loop (line 15). Only one token colored  $cl$  is available because the  $(\gamma - 1)$  other tokens colored  $cl$  were already added into the multiset  $tokens_i$  during its initialization at line 14.

As already said, when the predicate  $|tokens_i| < |to\_color_i|$  (line 15) becomes false,  $tokens_i$  contains enough colored tokens to assign to  $p_i$ ’s children. This assignment is done at lines 25-27. Let  $ch = |to\_color_i|$  (number of children of  $p_i$ );  $p_i$  extracts  $ch$  pairwise disjoint and non-empty subsets of the multiset  $tokens_i$ , and assigns each of them to a different neighbor. “Non-empty non-intersecting multisets” used at line 25 means that, if each of  $z$  multisets  $tk[id_x]$  contains a token with the same color, this colored token appears at least  $z$  times in the multiset  $tokens_i$ .

If the message  $COLOR(sender, cl\_map, -)$  received by  $p_i$  is not the first one, it was sent by one of its children. In this case,  $p_i$  keeps in its color set  $color\_map_i[id_i]$  ( $colors_i$ ) only the colors allowed by its child  $sender$  (line 30). Hence, when  $p_i$  has received a message  $COLOR()$  from each of its children, its color set  $colors_i$  has its final value.

## 6.5 Algorithm: broadcast of a message

A process  $p_i$  is allowed to broadcast a message only at the rounds corresponding to a color it obtained (a color in  $colors_i = color\_map_i[id_i]$  computed at lines 11, 17, and 30), provided that its current local state is 1 or 3 (line 31).

If  $state_i = 1$ ,  $p_i$  received previously a message  $COLOR()$ , which entailed its initial coloring and a proposal to color its children (lines 13-28). In this case,  $p_i$  propagates the tree traversal by broadcasting a message  $COLOR()$  (line 33), which will provide each of its children with a coloring proposal. Process  $p_i$  then progresses to the local waiting state 2.

If  $state_i = 3$ , the coloring of the tree rooted at  $p_i$  is terminated. Process  $p_i$  consequently broadcasts the message  $TERM(parent_i, id_i)$  to inform its parent of it. It also progresses from state 3 to state 4, which indicates its local termination (line 35).

## 6.6 Algorithm: reception of a message $TERM()$

When a process  $p_i$  receives such a message it discards it if it is not the intended destination (line 37). If the message is for it,  $p_i$  adds the sender’s identity to the set  $colored_i$  (line 38). Finally, if  $colored_i = neighbors_i$ ,  $p_i$  learns that the subtree rooted at it is colored (line 39). It follows that, if  $p_i$  is the root ( $parent_i = i$ ), it learns that the algorithm terminated. Otherwise, it enters state 3, that will direct it to report to its parent the termination of the coloring of its underlying subtree.

## 6.7 Solving F3C( $n, \gamma, K, 1$ ) in a tree

Algorithm 2 can be easily modified to solve F3C( $n, \gamma, K, 1$ ). When a process enters state 3 (at line 29 or line 41), it reduces  $color\_map_i[id_i]$  (i.e.,  $colors_i$ ) to obtain a singleton.

## 7 F3C( $n, \gamma, K, \geq 1$ ) IN A TREE: COST AND PROOF

The proof assumes  $n > 1$ . Let us remember that  $colors_i$  and  $color\_map_i[id_i]$  are the same local variable of  $p_i$ , and  $p_r$  denotes the dynamically defined root process.

### 7.1 Cost of the algorithm

Each non-leaf process broadcasts one message  $COLOR()$ , and each non-root process broadcasts one message  $TERM()$ . Let  $x$  be the number of leaves. There are consequently  $(2n - (x + 1))$  broadcasts. As  $\Delta \leq x + 1$  <sup>(6)</sup>, the number of broadcast is upper bounded by  $2n - \Delta$ .

$TERM()$  messages are of fixed length, while  $COLOR()$  message carry some fixed-length information, and the dictionary  $color\_map_i$ , which contains  $\Delta_i + 1$  entries. As a result, the length of messages is bounded by  $O(\Delta)$ .

Given an execution whose dynamically defined root is the process  $p_r$ , let  $d$  be the height of the corresponding tree. A recursive analysis of message patterns yields a time complexity of  $O(d \lceil \frac{\Delta}{\gamma} \rceil)$ .

### 7.2 Proof of the algorithm

The proof is decomposed into lemmas showing that the algorithm (a) is itself conflict-free and  $\gamma$ -collision-free, (b) terminates, and (c) associates with each process  $p_i$  a non-empty set  $colors_i$  satisfying the Conflict-freedom,  $\gamma$ -Collision-freedom and Efficiency properties defined in Section 4. To this end, a notion of *well-formedness* suited to  $COLOR()$  messages is introduced.

**Lemma 1.** *Algorithm 2 is conflict-free.*

6. Let  $p_i$  be the process that has  $\Delta$  as degree. If  $p_i$  is the root of the tree, the tree contains at least  $\Delta$  leaf processes. This is because each neighbor of  $p_i$  is either a leaf or the root of a subtree that has at least one leaf process. And if  $p_i$  is not the root of the tree,  $p_i$  possesses  $\Delta - 1$  children, and the number of leaf processes is at least  $\Delta - 1$  following a similar reasoning.

**Proof** The algorithm uses two types of message: `COLOR()` and `TERM()`. We first show conflict-freeness for `COLOR()` messages (if a process broadcasts a message `COLOR()`, none of its neighbors is broadcasting any message in the same round). Let us first notice that a process  $p_i$  broadcasts at most one message `COLOR()`, and one message `TERM()` (this is due to the guard  $state_i \in \{1, 3\}$ , line 31, and the fact that the broadcast of a message makes its sender progress to the waiting state 2 or 4). Moreover, let us make the following observations.

- Observation 1: The first message sent by any process is of type `COLOR()` (line 33).
- Observation 2: Except for the root process, a message `COLOR()` is always broadcast by a process after it received a message `COLOR()` (which triggers the execution of lines 5-30).
- Observation 3: Except for leaf processes, a message `TERM()` is always broadcast by a process after it received a message `TERM()` from each of its children (lines 36-41 and line 34).

Observations 1 and 2 imply that when the root process broadcasts its `COLOR()` message, none of its neighbors is broadcasting a message, and they all receive the root's `COLOR()` message without conflict. Let us now consider a process  $p_i$ , different from the root, which receives its first message `COLORk` (from its parent  $p_k$ ). Because there is no cycle in the communication graph (a tree), all the children of  $p_i$  ( $neighbors_i \setminus \{p_k\}$ ) are in state 0, waiting for their `COLOR()` message. Moreover, due to Observations 1 and 2, they will receive from  $p_i$  their message `COLOR()` without conflict. After sending its `COLOR()` message,  $p_i$ 's parent  $p_k$  remains in the waiting state 2 until it receives a `TERM()` message from all its children (lines 38-39), which include  $p_i$ . As a consequence,  $p_k$  is not broadcasting any message in the round in which it receives  $p_i$ 's `COLOR()` message, which is consequently received without conflict by all its neighbors.

As far as the messages `TERM()` are concerned we have the following. Initially, only a leaf process can broadcast a message, and when it does it, its parent is in the waiting state 2 (since it broadcasts a message `COLOR()` at line 33 and it must receive messages `TERM()` to proceed to state 3). Hence, a message `TERM()` broadcast by a leaf cannot entail conflict. Let us now consider a non-leaf process  $p_i$ . It follows from Observation 3 that  $p_i$  can broadcast a message `TERM()` only when its children are in state 4 (in which they cannot broadcast), and its parent (because it has not yet received a message `TERM()` from each of its children) is in the waiting state 2. Hence, we conclude that the broadcast of a message `TERM()` by a non-leaf process is conflict-free, which concludes the proof of the lemma.  $\square$  *Lemma 1*

**Definition** A message `COLOR(sender, cl_map, max_cl)` is *well formed* if its content satisfies the following properties. Let  $sender = id_i$ .

- M1 The keys of the dictionary data structure `cl_map` are the identities in  $neighbors_i \cup \{id_i\}$ .
- M2  $\forall id \in (neighbors_i \cup \{id_i\}) : cl\_map[id] \neq \emptyset$ .
- M3  $\forall id \in neighbors_i : cl\_map[id] \cap cl\_map[id_i] = \emptyset$ .
- M4  $\forall c : |\{id_j \in neighbors_i : c \in cl\_map[id_j]\}| \leq \gamma$ .
- M5  $1 < max\_cl \leq \lceil \frac{\Delta}{\gamma} \rceil + 1$ .
- M6  $\forall id \in (neighbors_i \cup \{id_i\}) : 0 \leq cl\_map[id] < max\_cl$ .

Once established in Lemma 3, not all properties M1-M6 will be explicitly used in the lemmas that follow. They are used by induction to proceed from one well-formed message to another one.

**Lemma 2.** *If a message `COLOR(sender, cl_map, max_cl)` received by a process  $p_i \neq p_r$  is well formed and entails the execution of lines 7-29, the **while** loop (lines 15-24) terminates, and, when  $p_i$  exits the loop, the sets  $colors_i$  and  $color\_map_i[parent_i]$  are not empty, and their intersection is empty.*

**Proof** Let us consider a process  $p_i \neq p_r$  that receives a well-formed `COLORj(sender, cl_map, max_cl)` message from  $p_j$ . Let us assume `COLOR()` causes  $p_i$  to start executing the lines 7-29, i.e., `COLOR()` is the first such message received by  $p_i$ . The body of the **while** loop contains two lines (lines 17 and 23) that select elements from two sets,  $colors_i$  and  $color\_map_i[parent_i]$  respectively.

Before discussing the termination of the **while** loop, we show that lines 17 and 23 are *well defined*, i.e. the sets from which the elements are selected are non-empty. To this aim, we prove by induction that the following invariant holds in each iteration of the loop:

$$color\_map_i[parent_i] \neq \emptyset, \quad (43)$$

$$colors_i \neq \emptyset, \quad (44)$$

$$|tokens_i| = \gamma \times max\_cl_i - \gamma \times |colors_i| - |color\_map_i[parent_i]|. \quad (45)$$

Just before the loop (i.e., before line 15), Assertion (43) follows from the assignment to  $color\_map_i[parent_i]$  at line 8 and the property M2 of `COLORj` ( $id_j = parent_i$ ). Assertion (44) also follows from M2 ( $colors_i$  is synonym of  $color\_map_i[id_i]$ ). Assertion (45) follows from M3, M6, and the initialization of  $max\_cl_i$  at line 12.

Let us now assume that Assertion (43) holds at the start of a loop iteration (i.e., just before lines 16). There are two cases.

- If  $|colors_i| > 1$ , lines 19-24 are not executed, and consequently  $color\_map_i[parent_i]$  is not modified. It follows from the induction assumption that Assertion (43) still holds.
- If  $|colors_i| \leq 1$ , we have the following. Because we are in the **while** loop, we have  $|tokens_i| < |to\_colors_i|$ , which, combined with Assertion (45), implies

$$|to\_colors_i| > \gamma \times max\_cl_i - \gamma \times |colors_i| - |color\_map_i[parent_i]|,$$

from which we derive

$$\begin{aligned} & |color\_map_i[parent_i]| \\ & > \gamma \times max\_cl_i - \gamma \times |colors_i| - |to\_colors_i|, \\ & > \gamma \times max\_cl_i - \gamma \times |colors_i| - (\Delta_i - 1), \\ & > \gamma \times \sigma_i - \gamma \times |colors_i| - (\Delta_i - 1), \\ & > \gamma \times (\lceil \frac{\Delta_i}{\gamma} \rceil + 1) - \gamma \times |colors_i| - (\Delta_i - 1), \\ & > \Delta_i + \gamma - \gamma \times |colors_i| - (\Delta_i - 1), \\ & > \gamma \times (1 - |colors_i|) + 1. \end{aligned}$$

Hence, because  $|colors_i| \leq 1$ , we obtain  $|color\_map_i[parent_i]| > 1$ , which means that  $p_i$ 's local variable  $color\_map_i[parent_i]$  contains at least two elements before the execution of line 16. Because only one color is removed from  $color\_map_i[parent_i]$ ,

this local variable remains non-empty after lines 23-24, thus proving Assertion (43).

Let us now assume that both Assertion (44) and Assertion (45) hold at the start of a loop iteration (i.e., just before line 16). There are two cases.

- Case  $|colors_i| > 1$ . In this case we have: (i) one color is removed from  $colors_i$ , (ii)  $\gamma$  colored tokens are added to  $tokens_i$ , and (iii)  $color\_map_i[parent_i]$  remains unchanged.  $|colors_i| > 1$  and (i) imply that Assertion (44) remains true; and (i) and (ii) mean that Assertion (45) is preserved.
- Case  $|colors_i| \leq 1$ . In this case we have: (i) one color is removed from  $color\_map_i[parent_i]$ , and one colored token added to  $tokens_i$ , and (ii)  $colors_i$  stays unchanged. (i) implies that Assertion (45) remains true, and (ii) ensures Assertion (44) by assumption.

This concludes the proof that the three assertions (43)–(45) are a loop invariant. Hence, Assertion (43) and Assertion (44) imply that lines 17 and 23 are well defined.

Let us now observe that, in each iteration of the loop, new colored tokens are added to  $tokens_i$ , and thus  $|tokens_i|$  is strictly increasing. Because  $|to\_color_i|$  remains unchanged, the condition  $|tokens_i| < |to\_color_i|$  necessarily becomes false at some point, which proves that the loop terminates.

Just after the loop, the invariant is still true. In particular Assertion (43) and Assertion (44) show that both the sets  $colors_i$  and  $color\_map_i[parent_i]$  are not empty when  $p_i$  exits the **while** loop.

Finally, due to the fact that the message  $COLOR_j()$  is well formed, it follows from M3 that we have  $colors_i \cap color\_map_i[parent_i] = \emptyset$  after line 11. As colors are added neither to  $colors_i$ , nor to  $color\_map_i[parent_i]$  in the loop, their intersection remains empty, which concludes the proof of the lemma.  $\square$  *Lemma 2*

**Lemma 3.** All messages  $COLOR()$  broadcast at line 33 are well formed.

**Proof** To broadcast a message  $COLOR()$ , a process  $p_i$  must be in local state 1 (line 33). This means that  $p_i$  executed line 28, and consequently previously received a message  $COLOR(sender, cl\_map, max\_cl)$  that caused  $p_i$  to execute lines 7-29.

Let us first assume that  $COLOR()$  is well formed. It then follows from Lemma 2 that  $p_i$  exits the while loop, and each of  $colors_i$  and  $color\_map_i[parent_i]$  is not empty (A), and they have an empty intersection (B). When considering the message  $COLOR(id_i, color\_map_i, max\_cl_i)$  broadcast by  $p_i$  we have the following.

- M1 follows from the fact that the entries of the dictionary data structure created by  $p_i$  are:  $color\_map_i[parent_i]$  (line 8),  $color\_map_i[id_i]$  (line 11), and  $color\_map_i[id]$  for each  $id \in to\_colors_i = neighbors_i \setminus \{parent_i\}$  (lines 10 and 27), and the observation that no entry is ever removed from  $color\_map_i$  is the rest of the code.
- M2 follows from (A) for  $color\_map_i[parent_i]$  and  $color\_map_i[id_i]$ , from line 25 for the identities in  $to\_colors_i = neighbors_i \setminus \{parent_i\}$  (due to  $|tokens_i| \geq$

$|to\_colors_i|$  when line 25 is executed, and the non-intersection requirement of the  $tk[id]$  sets, no  $tk[id]$  is empty), and from the observation that  $color\_map_i$  is not modified between the end of line 27 and the broadcast of line 33. This last claim is derived from the fact that  $color\_map_i$  is only modified when messages are received, and that neither  $p_i$ 's parent nor  $p_i$ 's children are in states that allow them to send messages while  $p_i$  is transitioning from line 27 to line 33.

- Similarly, M3 follows
  - for  $id = parent_i$ : from (B) and the fact that  $color\_map_i[parent_i]$  never increases,
  - for  $id \in to\_color_i = neighbors_i \setminus \{parent_i\}$ : from the fact that, due to lines 14 and 17, at line 25  $tokens_i$  contains no token whose color belongs to  $colors_i$ , from which we have  $tk[id] \cap color\_map_i[id_i] = \emptyset$  for any  $id \in to\_color_i$ .
- M4 follows from the construction of  $tokens_i$ . This construction ensures that, for any color  $c$ ,  $tokens_i$  contains at most  $\gamma$  tokens with color  $c$  (lines 14, 18, and 22).
- M5 is an immediate consequence of the assignment  $max\_cl_i \leftarrow \max(max\_cl, \sigma_i)$  at line 12.
- M6 follows from the following observations:
  - for  $id \in \{id_i, parent_i\}$ : from  $max\_cl \leq max\_cl_i$  (line 12) and the fact that the message  $COLOR()$  received by  $p_i$  is well formed (hence  $color\_map_i[id_i] \cup color\_map_i[parent_i] \subseteq [0, \dots, (max\_cl - 1)]$ ),
  - for  $id \in to\_color_i = neighbors_i \setminus \{parent_i\}$ : from the fact that  $tokens_i$  contains only tokens whose color is in  $[0, \dots, (max\_cl_i - 1)]$  (line 14).

The previous reasoning showed that, if a process receives a well formed message  $COLOR()$ , executes lines 7-29 and line 33, the message  $COLOR(id_i, color\_map_i, max\_cl_i)$  it will broadcast at this line is well formed. Hence, to show that all messages broadcast at line 33 are well formed, it only remains to show that the message  $COLOR(id_r, color\_map_r, max\_cl_r)$  broadcast by the root  $p_r$  is well formed. Let us remember that  $neighbors_r$  is a constant defined by the structure of the tree, and  $parent_r = id_r \notin neighbors_r$ .

Let us notice that  $COLOR(id, cl\_map, max\_cl)$  sent by  $p_r$  to itself at line 4 is not well formed. This is because,  $cl\_map[id]$  is not defined for  $id \in neighbors_i$ . When  $p_r$  receives this message we have the following after line 14:

$|tokens_r| = \gamma \times \sigma_r - \gamma = \gamma \times (\sigma_r - 1) = \gamma \lceil \frac{\Delta_r}{\gamma} \rceil \geq \Delta_r$ , from which we conclude  $|tokens_r| \geq \Delta_r = |to\_colors_r| = |neighbors_i|$ . Hence,  $p_r$  does not execute the loop body, and proceeds to lines 25-27 where it defines the entries  $color\_map_r[id]$  for  $id \in to\_colors_r = neighbors_r$ . A reasoning similar to the previous one shows that the message  $COLOR(id_r, color\_map_r, max\_cl_r)$  broadcast by  $p_r$  at line 33 satisfies the properties M1-M6, and is consequently well formed. (The difference with the previous reasoning lies in the definition of the set  $to\_colors_i$  which is equal to  $neighbors_i \setminus \{parent_i\}$  for  $p_i \neq p_r$ , and equal to  $neighbors_r$  for  $p_r$ .)  $\square$  *Lemma 3*

**Lemma 4.** If a process  $p_i$  computes a color set ( $colors_i$ ), this set is not empty.

**Proof** Let us first observe that, if a process  $p_i \neq p_r$  receives a message  $\text{COLOR}(-, cl\_map, -)$ , the previous lemma means that this message is well formed, and due to property M2, its field  $cl\_map[id_i]$  is not empty, from which follows that the initial assignment of a value to  $color\_map_i[id_i] \equiv colors_i$  is a non-empty set. Let us also observe, that, even if it is not well formed the message  $\text{COLOR}(-, cl\_map, -)$  received by the root satisfies this property. Hence, any process that receives a message  $\text{COLOR}()$  assigns first a non-empty value to  $color\_map_i[id_i] \equiv colors_i$ .

Subsequently, a color can only be suppressed from  $color\_map_i[id_i] \equiv colors_i$  at line 30 when  $p_i$  receives a message  $\text{COLOR}()$  from one of its children. If  $p_i$  is a leaf, it has no children, and consequently never executes line 30. So, let us assume that  $p_i$  is not a leaf and receives a message  $\text{COLOR}(id_j, cl\_map, -)$  from one of its children  $p_j$ . In this case  $p_i$  previously broadcast at line 33 a message  $\text{COLOR}(id_i, color\_map_i, -)$  that was received by  $p_j$  and this message is well formed (Lemma 3).

A color  $c$  that is suppressed at line 30 when  $p_i$  processes  $\text{COLOR}(id_j, cl\_map, -)$  is such that  $c \in colors_i$  and  $c \notin cl\_map[id_i]$ .  $cl\_map[id_i]$  can be traced back to the local variable  $color\_map_j[id_i]$  used by  $p_j$  to broadcast  $\text{COLOR}()$  at line 33. Tracing the control flow further back,  $color\_map_j[id_i]$  was initialized by  $p_j$  to  $color\_map_i[id_i]$  (line 8) when  $p_j$  received the well-formed message  $\text{COLOR}()$  from  $p_i$ . When processing the  $\text{COLOR}()$  message received from  $p_i$ , process  $p_j$  can suppress colors from  $color\_map_j[id_i]$  only at line 23, where it suppresses colors starting from the greatest remaining color. We have the following.

- If  $p_i$  is not the root, the message  $\text{COLOR}()$  it received was well formed (Lemma 3). In this case, it follows from the proof of Lemma 2 that it always remains at least one color in  $color\_map_j[id_i]$ .
- If  $p_i = p_r$ , its set  $colors_r$  is a singleton (it “received”  $\text{COLOR}(id_r, cl\_map_r, -)$  where  $cl\_map_r$  has a single entry, namely  $cl\_map_r[id_r] = \{1\}$ ). When  $p_j$  computes  $tokens_j$  (line 14) we have

$$|tokens_j| = \gamma \times \max(\sigma_r, \sigma_j) - \gamma = \gamma \lceil \frac{\max(\Delta_r, \Delta_j)}{\gamma} \rceil \geq \max(\Delta_r, \Delta_j) \geq \Delta_j = |to\_colors_j|,$$

from which follows that  $|tokens_j| \geq |to\_colors_j| = |neighbors_j| - 1$ . Hence,  $p_j$  does not execute the loop, and consequently does not modify  $color\_map_j[id_r]$ .

Consequently, the smallest color of  $colors_i \equiv color\_map_i[id_i]$  is never withdrawn from  $color\_map_j[id_i]$ . It follows that, at line 30,  $p_i$  never withdraws its smallest color from the set  $color\_map_i[id_i]$ .  $\square$  Lemma 4

**Lemma 5.** If  $p_i$  and  $p_j$  are neighbors  $colors_i \cap colors_j = \emptyset$ .

**Proof** As all color sets are initialized to  $\emptyset$ , the property is initially true. We show that, if a process receives a message  $\text{COLOR}()$ , the property remains true. As  $\text{TERM}()$  messages do not modify the coloring (lines 36-41) they do not need to be considered.

Let us consider two neighboring processes  $p_i$  and  $p_j$ , which compute their color sets (if none or only one of  $p_i$  and  $p_j$  computes its color set, the lemma is trivially satisfied). As the network is a tree, one of them is the parent of the other. Let  $p_i$  be the parent of  $p_j$ .

Process  $p_i$  broadcasts a message  $\text{COLOR}(-, cl\_map, -)$  at line 33 in which the set  $cl\_map[id_j]$  is  $color\_map_i[id_j]$ , as computed at line 25. If this message is received by  $p_j$ , this set will in turn be assigned to  $color\_map_i[id_j]$  at  $p_j$ . As this message is well formed (Lemma 3), we therefore have  $color\_map_i[id_i] \cap color\_map_j[id_j] = \emptyset$  (Property M3 of a well-formed message). Then, while  $p_i$  can be directed to suppress colors from  $color\_map_i[id_i]$  at line 30, it never adds a color to this set. The same is true for  $p_j$  and  $color\_map_j[id_j]$ . It follows that the predicate  $color\_map_i[id_i] \cap color\_map_j[id_j] = \emptyset$  can never be invalidated.  $\square$  Lemma 5

**Lemma 6.**  $\forall i, \forall c : |\{j : j \in neighbors_i \wedge c \in colors_j\}| \leq \gamma$ .

**Proof** The property is initially true. We show that it remains true when processes receive messages.

Let us consider a process  $p_i$  that broadcasts a message  $\text{COLOR}()$ . Due to the fact that such messages are broadcast only at line 33, it follows from Lemma 3 that the message  $\text{COLOR}(id_i, cl\_map, -)$  broadcast by  $p_i$  is well formed. Hence, it satisfies property M4. When processing this message

- A each child  $p_j$  of  $p_i$  adopts  $cl\_map[id_j]$  as its initial color set and assigns it to  $color\_map_j[id_j]$ ;
- B  $p_i$ 's parent  $p_k$  uses  $cl\_map[id_k]$  to update  $color\_map_k[id_k]$  at line 30 such that  $color\_map_k[id_k] \subseteq cl\_map[id_k]$ .

(A), (B), and M4 imply that just after  $p_i$ 's neighbors have processed  $p_i$ 's message, the lemma holds. As already seen in the proof of other lemmas,  $color\_map_j[id_j]$  may subsequently decrease, but never increases: colors can be suppressed from  $color\_map_j[id_j]$  (line 30) but never added to it. And the same is true at  $p_i$  for its set of colors  $color\_map_i[id_i]$ , and at its parent  $p_k$  for  $color\_map_k[id_k]$ . It then follows that  $|\{j : j \in neighbors_i \wedge c \in colors_j\}| \leq \gamma$  throughout the execution of the algorithm, which concludes the proof of the lemma.  $\square$  Lemma 6

**Lemma 7.** Algorithm 2 is  $\gamma$ -collision-free.

**Proof** We have to show that no process can have more than  $\gamma$  of its neighbors broadcasting during the same round. Initially, all processes are in state 0. Let us consider a process  $p_i$  and assume that one of its neighbors  $p_j$  is broadcasting a message. Let us further assume that this message is of type  $\text{COLOR}()$ .

- If  $p_j$  is  $p_i$ 's parent,  $p_j$ 's  $\text{COLOR}()$  message is the first message received by  $p_i$ , and both  $p_i$  and its children ( $p_i$ 's remaining neighbors) are in state 0, and hence silent. There is no collision at  $p_i$ .
- If  $p_j$  is one of  $p_i$ 's children, the value  $slot\_span_j$  used by  $p_j$  at line 31 is equal to  $max\_cl$  contained in the message  $\text{COLOR}(-, -, max\_cl)$  first received by  $p_j$  from  $p_i$ . Because of Lemma 3, this message is well formed, and consequently satisfies property M6. Any other child  $p_\ell$  of  $p_i$  broadcasting during this round will have received the same first message, and will therefore be using the same  $slot\_span_\ell = max\_cl$  value. It follows from Property M6, the assignment of line 11 executed by any child  $p_\ell$  (of  $p_i$ ) that received the message, and

the fact that its set  $colors_\ell$  can only decrease after being first assigned, that  $colors_\ell \subseteq [0, \dots, slot\_span_\ell - 1]$  for any child  $p_\ell$  of  $p_i$  (C).

Lemma 6, Property (C), and the *CLOCK*-based predicate defining the rounds at which a process is allowed to broadcast (line 31), imply that at most  $\gamma$  children of  $p_i$  can broadcast during the same round. If  $p_i$  has a parent  $p_k$  (i.e.  $p_i$  is not the root), both  $p_i$  and  $p_k$  are in state 2, and hence  $p_k$  is silent, proving the lemma. If  $p_i$  is the root, all its neighbors are its children, and the lemma also holds.

The same reasoning applies to the messages  $TERM()$  broadcast by the children of  $p_i$  and its parent.

□<sub>Lemma 7</sub>

**Lemma 8.** *Each process computes a set of colors, and the root process knows when coloring is terminated.*

**Proof** Let us first observe that, due to Lemmas 2 and 3, no process  $p_i \neq p_r$  can loop forever inside the **while** loop (lines 15-24), when it receives its first message  $COLOR()$ . The same was proved for the root  $p_r$  at the end of the proof of Lemma 4. Moreover, a process cannot block at line 30 when it receives other messages  $COLOR()$  (one from each of its children). Hence, no reception of a message  $COLOR()$  can prevent processes from terminating the processing of the message. The same is trivially true for the processing of a message  $TERM()$ .

Let us first show that each process obtains a non-empty set of colors. To this end, we show that each non-leaf process broadcasts a message  $COLOR()$ .

- When the root process  $p_r$  receives the external message  $START()$ , it “simulates the sending to itself” of the message  $COLOR(id_r, color\_map_r, \sigma_r)$ , where the dictionary data structure  $color\_map_r$  has a single element, namely,  $color\_map_r[id_r] = \{1\}$  (line 11). The root  $p_r$  executes consequently the lines 7-29, during which it obtains a color, and computes a set of proposed colors  $color\_map_r[id_j]$  for each of its children  $p_j$  (lines 25-27). It then progresses to the local non-waiting state 1 (line 28). Hence, during the first round, it broadcasts to its neighbors the message  $COLOR(id_r, color\_map_r, \sigma_r)$ . Because the algorithm is conflict- and collision-free (Lemmas 1 and 5), this message is received by all the root’s neighbors.
- Let us now consider a process  $p_i$  that receives a message  $COLOR(sender, color\_map, max\_cl)$  for the first time. It follows from Lemma 4 that  $p_i$  starts computing a non-empty set  $colors_i$  and enters the waiting state 1 (line 28). Finally, as  $colors_i \subseteq [0, \dots, slot\_span_i - 1]$ , and *CLOCK* never stops increasing, the predicate of line 31 is eventually satisfied. It follows that  $p_i$  broadcasts the message  $COLOR(id_i, color\_map_i, max\_cl_i)$ . As above all of  $p_i$ ’s neighbors will receive this message. It follows that  $COLOR()$  messages flood the tree from the root to the leaves. Moreover, when a process  $p_i$  has received a message  $COLOR()$  from each of its neighbors (children and parent), it has obtained the final value of its color set  $color\_map_i[id_i] = colors_i$ . Due to lemma 4, this set is not empty, which concludes the first part of the proof.

Let us now show that the root detects coloring termination. This relies on the messages  $TERM()$ . As previously, due to Lemma 1 and Lemma 7, these messages entail neither message conflicts nor message collisions.

Let us observe that each leaf process eventually enters the non-waiting state 3. When the predicate of line 31 is satisfied at a leaf  $p_\ell$  (this inevitably occurs), this process broadcasts the message  $TERM()$  to its parent  $p_i$ . Then, when  $p_i$  has received a message  $TERM()$  from each of its children, it broadcasts  $TERM()$  to its own parent. This sequence repeats itself on each path from a leaf to the root. When the root has received a message  $TERM()$  from each of its children, it learns termination (line 40), which concludes the proof of the lemma.

□<sub>Lemma 8</sub>

**Lemma 9.**  $|\bigcup_{1 \leq i \leq n} colors_i| = \lceil \frac{\Delta}{\gamma} \rceil + 1$ .

**Proof** Let  $p_r, p_a, \dots, p_\ell$  be a path in the tree starting at the root  $p_r$  and ending at a leaf  $p_\ell$ . It follows from

- the content of the parameter  $max\_cl$  of the messages  $COLOR(sender, cl\_map, max\_cl)$  that are broadcast along this path of the tree (broadcast at line 33 and received at line 5), and
- the assignment of  $\max(max\_cl, \sigma_i)$  to  $max\_cl_i$  at line 12,

that  $max\_cl_\ell = \max(\sigma_r, \sigma_a, \dots, \sigma_\ell)$ . Let  $p_{\ell 1}, \dots, p_{\ell x}$  be the set of leaves of the tree. It follows that  $\max(max\_cl_{\ell 1}, \dots, max\_cl_{\ell x}) = \max(\sigma_1, \dots, \sigma_n)$ , i.e., the value  $max\_cl$  carried by any message is  $\leq \lceil \frac{\Delta}{\gamma} \rceil + 1$ .

The fact that a process  $p_i$  uses only colors in  $[0, \dots, (max\_cl_i - 1)]$ , combined with Theorem 1 implies the lemma. The algorithm is consequently optimal with respect to the number of colors.

□<sub>Lemma 9</sub>

**Theorem 4.** *Let  $K = \lceil \frac{\Delta}{\gamma} \rceil + 1$ . Algorithm 2 is a  $C2\gamma$ -free algorithm, which solves  $F3C(n, \gamma, K, \geq 1)$  in tree networks. Moreover, it is optimal with respect to the value of  $K$ .*

**Proof** The proof that Algorithm 2 terminates follows from Lemma 6. The proof that it is  $C2\gamma$ -free follows from Lemma 1 and Lemma 7. The proof that it satisfies the Conflict-freedom, Collision-freedom, and Efficiency properties defining the  $F3C(n, \gamma, K, \geq 1)$  problem follows from Lemmas 2-6, and Lemma 9. The proof of its optimality with respect to  $K$  follows also from Lemma 9.

□<sub>Theorem 4</sub>

## 8 EXPERIMENTAL EVALUATION

In this section, we evaluate experimentally our protocol using the OMNeT++ network simulator (version 5.3) [27], and compare its performance against that of the DRAND protocol [25], a reference distributed TDMA slot allocation algorithm. We have made the code we have used for our experiments publicly available on-line<sup>7</sup>.

### 8.1 The DRAND algorithm

DRAND uses a greedy color allocation strategy implemented with a fully decentralized control. It assumes a

7. <https://gitlab.inria.fr/WIDE/f3c-evaluation>

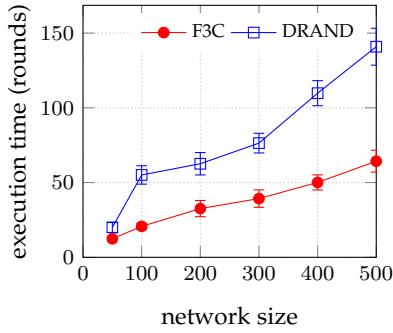


Figure 2. Execution time (measured in rounds) of our F3C-algorithm and DRAND. On average, our solution is more than twice faster ( $\times 2.11$ ).

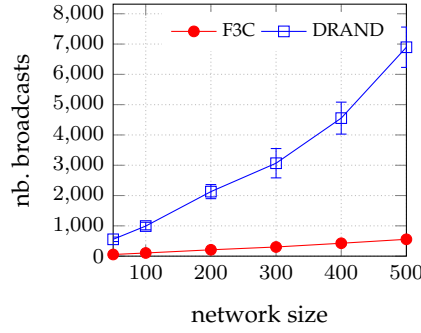


Figure 3. Number of broadcasts performed by our F3C-algorithm and DRAND. On average, our algorithm uses 10 times fewer broadcasts.

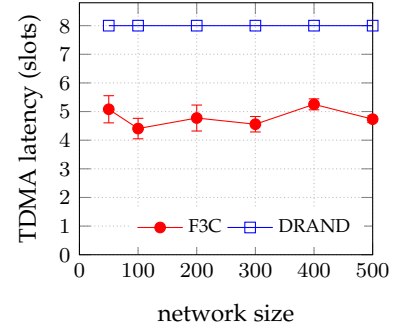


Figure 4. Quality of the resulting TDMA schedule. On average, our algorithm yields latency values that are 40% lower.

MAC (*Medium Access Control*) protocol to detect and avoid conflicts, and includes adaptive re-transmission mechanisms to tolerate collisions.

While in its initial state ('IDLE'), a DRAND node  $p$  tosses an even coin. If  $p$  gets head, it runs a lottery with a  $\frac{1}{k}$  chance of success, where  $k$  is the number of uncolored nodes in  $p$ 's 2-hop neighborhood. If  $p$  wins the lottery, it initiates a 2-phase commit negotiation with its neighbors in order to select an available time slot. This negotiation might fail if some neighbors are already involved in another 2-phase commit negotiation, in which case  $p$  (the requesting node) returns to its IDLE state.

Because of its stochastic activation mechanism, and because the 2-phase commit phase may fail, DRAND is probabilistic by construction. It also does not avoid collisions while executing. For a conservative comparison, we have ignored collisions when simulating DRAND (we have let colliding broadcasts reach their destination), and have assumed that the underlying MAC protocol was perfect (thus avoiding conflicts).

## 8.2 Target network trees

We exercise both algorithms on randomly generated trees that exhibit a predetermined  $\Delta$  (the maximum node degree), depth  $d$ , and size  $n$ . In the rest of the evaluation  $\Delta$  is fixed at 7, and depth at 6. The trees are constructed recursively from the root node, choosing neighborhood sizes uniformly between 1 and  $\Delta$  until a size of  $n$  nodes is reached. We then prune trees whose maximum degree does not reach  $\Delta$ . Because DRAND is not designed to handle multiple channels, we assume  $\gamma = 1$  (only one channel is available) for a fair comparison.

## 8.3 Metrics

We evaluate our protocol and DRAND in terms of the following metrics:

- **Execution time** is the time each algorithm needs to allocate colors to all nodes in the network, measured in rounds.
- **Broadcast complexity** is the total number of broadcast operations performed by each algorithm.
- **TDMA latency** assesses the quality of the resulting coloring when interpreted as a TDMA schedule. TDMA

latency measures the average number of TDMA slots a node must wait before it is allowed to broadcast again.

## 8.4 Results

Figures 2 (Execution time), 3 (Broadcast complexity), and 4 (TDMA latency) show the results we obtain for network sizes varying from 50 to 500 nodes. In Figures 2 and 3 each point is averaged over 10 experiments, in Figure 4 over 20 experiments. Error bars show 95%-level confidence intervals computed using Student's test statistics.

On all metrics, our F3C-algorithm clearly outperforms DRAND. DRAND is particularly hampered by its probabilistic nature, which slows it down (Figure 2), and causes it to use about ten times more messages than we do (Figure 3). In terms of schedule quality, both algorithms use the optimal number of colors/slots,  $K = \lceil \frac{\Delta}{\gamma} \rceil + 1 = \frac{7}{1} + 1 = 8$ . However, because our F3C-algorithm is able to assign (when possible) several colors/slots to individual nodes, F3C-nodes do not necessarily need to wait the repetition of the TDMA schedule to communicate again, yielding a decrease in latency of about 40% on average across all network sizes.

## 9 CONCLUSION

In this paper we investigated the problem of constructing a  $\gamma$ -frugal vertex coloring using a distributed algorithm experiencing *conflicts* and *collisions*. This problem arises in particular when assigning rounds (slots) to processes (nodes) in broadcast/receive TDMA wireless multi-channel networks in which each node only has access to one transceiver.

We presented a deterministic, distributed, parallel, color-optimal, collision- and conflict-free algorithm which solves this distributed vertex-coloring problem for tree networks. This algorithm only uses  $K = \lceil \frac{\Delta}{\gamma} \rceil + 1$  colors (where  $\Delta$  is the maximal degree of the graph), and is optimal with respect to the total number of colors that can be used. We have further evaluated the performance of this algorithm using simulations. The obtained experimental results show our solution clearly outperforms a reference protocol for distributed TDMA slot-allocation.

Moreover, from an algorithmic point of view, the proposed algorithm is versatile, making it an attractive starting point to address other related problems. For instance, in a heterogeneous network, lines 25-27 could be modified to



take into account additional constraints arising from the capacities of individual nodes, such as their ability to use only certain frequencies.

Last but not least, a major challenge for future work consists in solving the F3C problem in general graphs using a parallel rather than sequential deterministic algorithm. The new difficulty is then to take into account cycles in a distributed setting in which the global graph topology is not known beforehand, but must be discovered on the fly while avoiding conflicts and collisions.

## ACKNOWLEDGMENTS

This work has been partially supported by the French ANR project DESCARTES (devoted to abstraction layers in distributed computing), and the Franco-Hong Kong ANR-RGC Joint Research Programme 12-IS02-004-02 CO2Dim.

## REFERENCES

- [1] Angluin D., Local and global properties in networks of processors. *12th ACM Symposium on Theory of Computation*, pp. 82–93 (1981)
- [2] Attiya H. and Welch J., *Distributed computing: fundamentals, simulations and advanced topics*, (2d Edition), Wiley-Interscience, 414 pages (2004)
- [3] Barenboim L. and Elkin M., *Distributed graph coloring, fundamental and recent developments*, Morgan & Claypool Publishers, (2014)
- [4] Barenboim L., Elkin M., and Kuhn F., Distributed (Delta+1)-coloring in linear (in Delta) time. *SIAM Journal of Computing*, 43(1):72–95 (2014)
- [5] Blair J. and Manne F., An efficient self-stabilizing distance-2 coloring algorithm. *Proc. 16th Colloquium on Structural Information and Communication Complexity (SIROCCO'10)*, pp. 237–251 (2009)
- [6] Cowen, L., Goddard W., and Jesurum, C. E. Defective coloring revisited. *Journal of Graph Theory*, 205–219, (1997)
- [7] Chipara O., Lu C., Stankovic J., and Roman. G.-C., Dynamic conflict-free transmission scheduling for sensor network queries. *IEEE Transactions on Mobile Computing*, 10(5):734–748 (2011)
- [8] Chung K.-M., Pettie S., Su H.-H. Distributed Algorithms for the Lovász Local Lemma and Graph Coloring. *Proc. 33th ACM Symposium Principles of Distributed Computing*, 2014
- [9] Frey D., Lakhlef H., and Raynal M., Optimal collision/conflict-free distance-2 coloring in wireless broadcast/receive synchronous tree networks. *Proc. 45th Conf. on Parallel Processing*, pp. 350–359 (2016)
- [10] Gairing M., Goddard W., Hedetniemi S. T., Kristiansen P., and McRae A. A., Distance-two information in self-stabilizing algorithms. *Parallel Processing Letters*, 14(03-04):387–398, (2004).
- [11] Herman T., Tixeuil S., A distributed TDMA slot assignment algorithm for wireless sensor networks. *Proc. Int'l Workshop on Algorithmic Aspects of Wireless Sensor Networks*, pp. 45–58 (2004)
- [12] Hugh Hind H., Molloy M., and Reed B., Colouring a graph frugally. *Combinatorica*, 17(4):469–482, (1997).
- [13] Jemili I., Ghrab D., Belghith A., Derbel B., and Dhraief A., Collision aware coloring algorithm for wireless sensor networks. In *Proc. 9th Int'l Wireless Communication and Mobile Computing Conference (IWCMC'13)*, pages 1546–1553 (2013)
- [14] Kang, R. J., and Müller T. Frugal, acyclic and star colourings of graphs. *Discrete Applied Mathematics* 159.16 (2011): 1806–1814.
- [15] Lakhlef H., Raynal M., and Taïani F., Vertex Coloring with Communication and Local Memory Constraints in Synchronous Broadcast Networks. *12th International Symposium on Algorithms and Experiments for Wireless Sensor Networks*, Denmark (2016)
- [16] Lynch N.A., *Distributed algorithms*. Morgan Kaufmann, 872 pages (1996)
- [17] Mahfoudh S., Chalhoub G., Minet P., Misson M., and Amdouni I., Node coloring and color conflict detection in wireless sensor networks. *Future Internet*, 2(4):469, 2010.
- [18] Molloy, M., and Bruce R., *Graph colouring and the probabilistic method*. Vol. 23. Springer (Algorithms and combinatorics), 2002.
- [19] Molloy, M., Reed, B.: *Asymptotically optimal frugal colouring* J. Comb. Theory Ser. B100 (2), 226–246 (2010)
- [20] Narayanan L., Channel assignment and graph multicoloring. *Handbook of wireless networks and mobile computing*, John Wiley & Sons, ISBN: 0-471-41902-8, 2002
- [21] Peleg D., *Distributed computing, a locally sensitive approach*. SIAM Monographs on Discrete Mathematics and Applications, 343 pages, ISBN 0-89871-464-8 (2000)
- [22] Raychaudhuri, A. Further results on T-coloring and frequency assignment problems. *SIAM Journal on Discrete Mathematics* 7.4 (1994): 605–613.
- [23] Raynal M., *Fault-tolerant agreement in synchronous message-passing systems*. Morgan & Claypool Publishers, (ISBN 978-1-60845-525-6) (2010)
- [24] Raynal M., *Distributed algorithms for message-passing systems*. Springer, 500 pages, ISBN 978-3-642-38122-5 (2013)
- [25] I. Rhee, A. Warrior, J. Min, and L. Xu, *Drand: Distributed randomized TDMA scheduling for wireless ad-hoc networks*. *IEEE Transactions on Mobile Computing*, V.: 8, pp:1384–1396, 2009
- [26] Zhang, X., Hong, J., Zhang, L., Shan, X., and Li, V. O. , CC-TDMA: Coloring and coding-based multi-channel TDMA scheduling for wireless ad hoc networks. *IEEE Wireless Communications and Networking Conference (WCNC)*, 2007
- [27] <https://omnetpp.org/>



CON 2015, IEEE UIC 2016 and ICCS 2018.

**Hicham Lakhlef** has been an Associate Professor (Maître de Conférence) at the University of Technology of Compiègne (UMR CNRS 6174) in France since 2016. Prior to this, he was a post-doctoral researcher at the Université de Rennes 1 - IRISA (UMR CNRS 6074). He obtained his Ph.D. degree from the University of Franche-Comté in 2014 in France. He obtained his Master's degree from the University of Picardie Jules Verne in 2011 in France. He served as a PC member for several conferences as IEEE EURO-



twelve books devoted to parallelism, distributed algorithms and systems (published by MIT Press, Wiley, Morgan & Claypool and Springer).

**Michel Raynal** has been a professor of *Informatics* since 1981 (Sup Telecom Brest 1981–83, and then University of Rennes 1). Since 2013 he is also Chair Professor at the Polytechnic University of Hong Kong. His main interest lies in the fundamental principles that underlie the design and the construction of distributed computing. Professor Michel Raynal has published more than 150 papers in international journals (from JACM to IEEE Computer), and more than 300 papers in conferences. He has also written



on-line social networks, data centers), with a focus on resilience, concurrency, and self-organization.

**François Taïani** has been a Professor in Distributed Computer Systems at the University of Rennes 1 and IRISA/Inria in Brittany, France since 2012. Prior to that, he was with Lancaster University (UK) from 2005 to 2012. François holds a PhD from Université Toulouse III (2004), a Diplom der Informatik from Universität Stuttgart (1998), and a Diplôme d'Ingénieur from Ecole Centrale Paris (France). His main interest lies in the scalability and programmability of complex distributed systems (e.g. overlays,